Révisions de PCSI : Compétences numériques

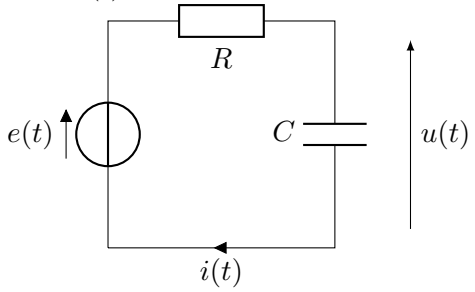
Table des matières

1 Méthode d'Euler pour résoudre un circuit du premier ordre

→ Circuit linéaire du premier ordre : *mettre en œuvre la méthode d'Euler pour simuler la réponse d'un système linéaire du premier ordre à une excitation de forme quelconque.*

1.1 Principe de la méthode d'Euler

On souhaite déterminer la réponse d'un circuit du premier ordre à une excitation de forme quelconque. Considérons un circuit RC série alimenté par une tension $e(t)$ d'après le montage suivant :



La tension $e(t)$ peut être éventuellement constante, constante par morceaux (par exemple un échelon de tension), sinusoïdale, etc. La tension u aux bornes du condensateur suit l'équation différentielle suivante sur l'intervalle de temps

$[t_0, t_f]$:

$$\frac{du}{dt} + \frac{u}{\tau} = \frac{e(t)}{\tau} \tag{1}$$

Cette équation se résout directement si $e(t)$ est constante par morceau (par exemple échelon de tension). Ou se résout en passant en complexe si $e(t)$ est sinusoïdale. Mais pour $e(t)$ quelconque, on peut utiliser une résolution numérique approchée de cette équation différentielle du premier ordre : la méthode d'Euler.

Pour la mettre en équation, utilisons la définition de la dérivée comme limite du taux d'accroissement, ainsi : $\frac{du}{dt} \simeq \frac{u(t+h)-u(t)}{h}$, avec h le pas de discrétisation du temps.

Ainsi :

$$\frac{du}{dt} = -\frac{u(t)}{\tau} + \frac{e(t)}{\tau} \tag{2}$$

$$\frac{u(t+h)-u(t)}{h} = -\frac{u(t)}{\tau} + \frac{e(t)}{\tau} \tag{3}$$

$$u(t+h) = u(t) + h \times \frac{f(t)-u(t)}{\tau} \tag{4}$$

On définit la suite $(t_i)_{i\in[\![0,n]\!]}$ des $n+1$ instants de calcul avec $t_i = t_0 + i \times h$ (donc $t_{i+1} = t_i + h$).

On note u_i la valeur approchée déterminée par la méthode d'Euler de u à l'instant t_i. À partir de l'expression précédente, on peut écrire :

$$\boxed{u_{i+1} = u_i + h \times \frac{e(t_i) - u_i}{\tau}} \tag{5}$$

Ainsi, connaissant la fonction e et la condition initiale $u_0 = u(t_0)$, on peut déterminer $u_1 = u_0 + h \times \frac{e(t_0)-u_0}{\tau}$. Puis $u_2 = u_1 + h \times \frac{e(t_1)-u_1}{\tau}$, etc.

1.2 Réponse en régime libre

Considérons le cas $R = 1$ kΩ, $C = 3.10^{-7}$ F dans le cas du régime libre $e(t) = 0$. On souhaite une étude entre $t_0 = 0$ et $t_f = 3$ ms (remarquer une durée d'étude grande devant τ), avec $n = 1000$ itérations. La condition initiale est $u_0 = 5$ V.

1. Compléter la ligne calculant le pas `h`, l'initialisation de `u`, la création de la liste des instants `Lt`.
2. Compléter les deux lignes dans la boucle `for`.

```
import matplotlib.pyplot as plt # pour les graphes
## parametres du circuit
R = 10**3                  # en Ohm
C = 3*10**-7               # en Farad
tau = R*C                  # pratique pour les calculs

## definition de l'intervalle d'etude
t0 = 0                     # instant initial
tf = 0.003                 # instant final
n = 1000                   # nombre d'iterations
h =                        # a completer : pas de calculs

## definition de la tension d'excitation
# cas d'un regime libre
def e(t):
    return 0

## initialisations
u =                        # a completer : condition initiale
Lu = [u]                   # a completer : initialisation de la liste des tensions
Lt =                       # a completer : liste des n+1 instants

## boucle methode Euler
for i in range(n):         # il reste n instants a determiner (i allant de 0 a n-1)
    u =                    # a completer : calcul de la valeur suivante
                           # a completer : ajout u a la liste des tensions

## Graphe
```

```
Le = [e(t) for t in Lt]
plt.plot(Lt,Lu, label = 'u(t)')            # representation de u(t)
plt.plot(Lt,Le, label = 'e(t)')            # representation de u(t)
plt.title('Reponse du circuit en fonction du temps')
plt.xlabel('t(s)')
plt.ylabel('tension(V)')
plt.legend()
plt.show()
```

1.3 Réponse à un signal en dents de scie

Cette fois le signal est une dent de scie de période $T = 1$ ms, cf figure 1. C'est une fonction affine valant $-a$ en $t = 0$ et $+a$ en $t = T$ avec $a = 2$ V, puis est périodique. Il suffit donc de définir $e(t)$ entre les instants 0 et T.

On utilisera `floor(x)` du module `math` qui renvoie la partie entière de `x`.

1. Soit $t > T$. On note $t_{\text{translat}} \in [0, T]$ l'instant tel que $e(t) = e(t_{\text{translat}})$. Exprimer t_{translat} en fonction de t et T en utilisant la partie entière. Compléter la ligne correspondant du code ci-dessous.
2. Compléter la ligne de retour de fonction en utilisant `t_translat`, `a` et `T`.

```
# cas d'une dent de scie periodique avec e(0)=-a et e(T)=a
def e(t):
    a = 2                          # amplitude en V
    T = 0.001                      # periode en s
    t_translat =                        # a completer : pour se ramener dans [0,T]
    return                              # a completer : valeur du signal
```

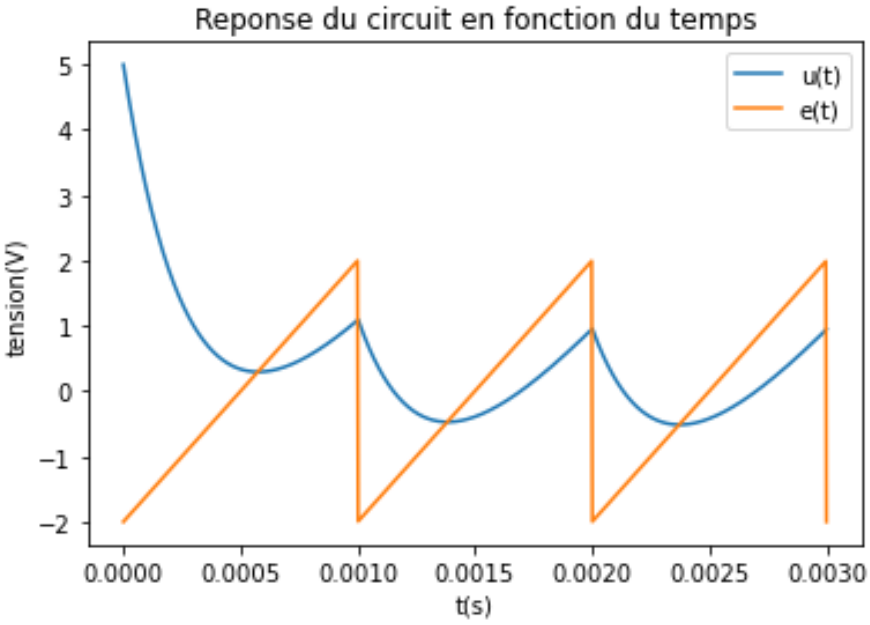


FIGURE 1 – Réponse du circuit à l'excitation en dent de scie.

2 Filtrage linéaire d'un signal périodique

→ Filtrage linéaire : *simuler l'action d'un filtre sur un signal périodique dont le spectre est fourni. Mettre en évidence l'influence des caractéristiques du filtre sur l'opération de filtrage.*

2.1 Calcul du filtrage d'un signal périodique non sinusoïdal

Considérons un signal d'entrée $e(t)$ périodique de fréquence f. On souhaite calculer le signal de sortie $s(t)$ à travers un filtre linéaire de fonction de transfert $\underline{H}(\omega)$. Le principe général est d'appliquer la fonction de transfert harmonique à chaque composante de $e(t)$.

⋆ On écrit la décomposition de Fourier du signal d'entrée :

$$e(t) = e_0 + \sum_{n=1}^{+\infty} e_n \cos\left(nft + \varphi_n\right) \tag{6}$$

⋆ Puis pour chaque composante n, on multiplie l'amplitude par le gain à fréquence $f_n = nf$: $G(f_n) = |\underline{H}(f_n)|$, et on ajoute à sa phase $\arg(\underline{H}(f_n))$:

$$s(t) = e_0 \times G(0)\cos(\arg(\underline{H}(0))) + \sum_{n=1}^{+\infty} e_n \times G(nf)\cos\left(nft + \varphi_n + \arg(\underline{H}(nf))\right) \tag{7}$$

Ainsi connaissant la fonction de transfert harmonique et/ou le diagramme de Bode d'un filtre linéaire, on peut connaître la réponse du filtre à n'importe quel signal périodique.

2.2 Étude de la simulation

Commençons par définir la fonction de transfert du filtre. Le nombre complexe noté i en maths et parfois j en physique s'obtient avec python par `1j`. Pour saisir $2j$, il faut donc taper `2*1j`.

1. Compléter le code suivant qui renvoie la valeur complexe de la fonction de transfert d'un basse-bas d'ordre 1 $H(\omega) = \frac{H_0}{1+jf/f_c}$.

```
## Bibliotheques necessaires
import matplotlib.pyplot as plt # pour les graphes
import numpy as np # pour manipuler les tableaux
from cmath import * # pour manipuler les complexes : abs pour le module et phase pour l'argument

## Definition de la fonction de transfert
def passe_bas(H0,fc,f):  # Passe bas ordre 1 de gain statique H0 et frequence de coupure fc
    return                                  # a completer
```

On crée maintenant la fonction `gene_signal` qui génère un signal connaissant son spectre en amplitude et en phase, c'est-à-dire les fréquences, amplitudes et phases à l'origine des temps des harmoniques.

Elle prend quatre arguments en entrée, trois listes de même longueur, et une autre liste :

— `frequence` : liste qui contient les fréquences des composantes du signal,

— `amplitude` : liste qui contient les amplitudes des composantes du signal (dans le même ordre que fréquence),

— `phase` : liste qui contient les phases à l'origine des temps des composantes du signal ,

— `t` : liste des instants où on veut calculer la valeur du signal.

Pour exprimer les signaux, on utilise le module `numpy` qui permet, notamment, de manipuler des tableaux (fonctionnement très proche des listes) qui sont parfois plus pratique à manier. Par exemple pour obtenir le tableau T qui contient $[\cos(1), \cos(3), \cos(-2)]$, on peut saisir :

```
L = [1, 3, -1]
T = np.cos(L)
```

La deuxième ligne va directement appliquer cos (avec la fonction `cos` du module `numpy`) à toutes les valeurs de la liste `L`, et renverra [1] un tableau qui contient les valeurs de $\cos(X)$.

2. Compléter le `range` de la boucle `for` dans le code suivant.
3. Compléter la génération du signal dans la boucle.

```
# frequence, amplitude et phase sont les tableaux contenant respectivement les infos des composantes du signal
# t est le tableau des instants ou on veut calculer le signal
def gene_signal(frequence,amplitude,phase,t):
    s = np.zeros(len(t))  # intialisation du signal : tableau de 0
    # Somme sur toutes les harmoniques voulues : 2 lignes a completer
    for i in range(
        s +=
    return s
```

Voici la suite du code permettant la comparaison du signal d'entrée et de sortie :

1. Avec une liste, on aurait écrit par exemple : `T = [np.cos(x) for x in L]`.

```python
## Generation d'un signal d'entree en creneau
f0 = 5          # frequence du fondamental du creneau en Hz
A = 2           # amplitude du creneau en V
n = 100         # nombre d'harmoniques souhaitees

# liste des frequences pour n composantes
freq = [(2*i - 1) *f0 for i in range(1,n+1)]
# liste des amplitudes pour n composantes
ampl = [4*A /((2*i -1) *pi) for i in range(1,n+1)]
# liste des phase a l'origine pour n composantes
phie = [ pi/2 for i in range(1,n+1) ]

# instants de calculs
Nt = 100000                     # nb de pts dans le domaine temporel
t = np.linspace(0,3/f0,Nt)      # sur trois periodes

# generation du creneau
entree = gene_signal(freq,ampl,phie,t)

## Determination du signal de sortie apres filtrage
sortie = np.zeros(Nt)           # initialisation

# filtre passe-bas de frequence de coupure fc et gain statique H0
H0 = 1
fc = 1
# Appliquer le filtre a chque composante de frequence f_i :
for i in range(len(freq)):
    H = passe_bas(H0,fc,freq[i])            # calcul de la fonction de transfert en f_i
    G = abs(H)                              # calcul du gain a f_i
    phi = phase(H)                          # calcul du dephasage a f_i
    # ajout de la composante filtree
    sortie = sortie + gene_signal([freq[i]],[ampl[i]*G],[phie[i] + phi],t)

# graphe
plt.plot(t,entree, label =' entree ')
plt.plot(t,sortie, label = 'sortie ')
plt.legend(loc='lower right')
plt.grid()
plt.xlabel('t (s)')
plt.ylabel('tension (V)')
plt.show()
```

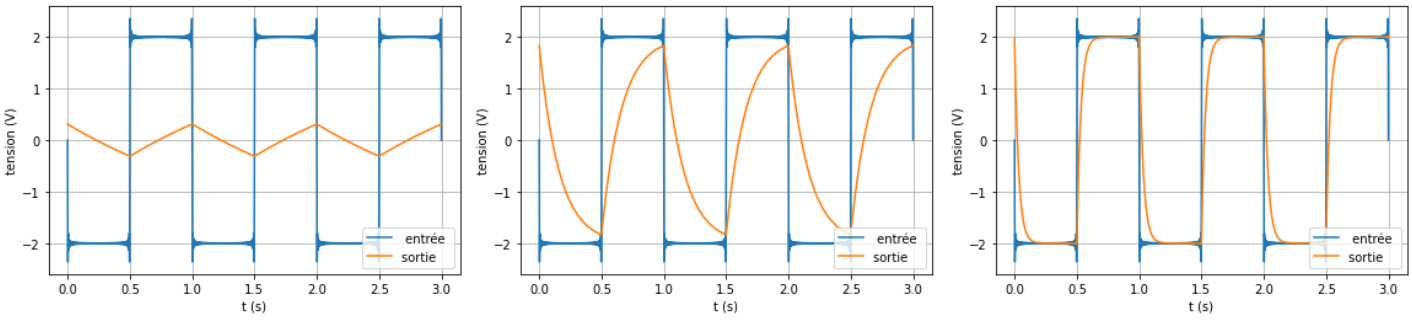


FIGURE 2 – Action d'un passe-bas du premier ordre sur un créneau pour $f_c = f/10$, $f_c = f$, $f_c = 5f$.

4. Interpréter la forme quasiment triangulaire du signal de sortie pour $f_c = f/10$.
5. Interpréter pourquoi même avec $f_c = 5f$, le signal de sortie n'est pas fidèle au signal d'entrée.

3 Résolution d'une équation différentielle d'ordre 2

→ Petits mouvements au voisinage d'une position d'équilibre stable : *résoudre numériquement une équation différentielle du deuxième ordre nonlinéaire et faire apparaître l'effet des termes nonlinéaires.*

Considérons un pendule pesant sans frottement dont la position est repérée par l'angle θ avec la verticale descendante. Son évolution est régie par l'équation différentielle :

$$\frac{d^2\theta}{dt^2} + \omega_0^2 \sin(\theta) = 0 \tag{8}$$

Avec ω_0 la pulsation propre. On note les conditions initiales $\theta_0 = \theta(t=0)$ et $\dot{\theta}_0 = \dot{\theta}(t=0)$. Cette équation non-linéaire ne peut pas être résolue analytiquement. Pour obtenir une solution, il faut effectuer un calcul approché :

— soit calcul analytique dans un cadre restreint : sous l'approximation des petits angles ($\theta \ll 1$), on peut calculer $\theta(t) = \theta_0 \cos(\omega_0 t) + \dot{\theta}_0 \sin(\omega_0 t)$. Dans ce cas, l'oscillation est sinusoïdale à pulsation ω_0.

— soit calcul numérique dans le cas général : c'est l'enjeu de la suite de ce texte.

3.1 Principe de la méthode d'Euler pour ED d'ordre 2

Pour résoudre une ED d'ordre 2 en θ, il suffit d'appliquer la méthode d'Euler au couple $\left(\dot{\theta};\theta\right)$ qui vérifie le système différentiel suivant dont on déduit le schéma d'Euler :

$$\begin{cases} \dfrac{d\dot{\theta}}{dt} = -\omega_0^2 \sin(\theta) \\ \dfrac{d\theta}{dt} = \dot{\theta} \end{cases} \quad \Rightarrow \quad \begin{cases} \dot{\theta}_{i+1} = \dot{\theta}_i - \omega_0^2 \sin(\theta_i) \cdot dt \\ \theta_{i+1} = \theta_i + \dot{\theta}_i \cdot dt \end{cases}$$

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

## Parametres et CI
w0 = 2            # pulsation propre en s**-1
theta0 = 1        # angle initial en rad
thetapoint0 = 0   # vitesse angulaire initiale en s**-1
t0 = 0                  # instant initial
tf = 30                 # instant final
N = 10000               # nombre d'iterations
t = np.linspace(t0, tf, N+1)       # instants
dt = (tf-t0)/N          # pas de temps

## Resolution par methode d'Euler
theta = [theta0]
thetapoint = [thetapoint0]
for i in range(N):
    thetapoint.append(thetapoint[i] - dt*np.sin(theta[i])*w0**2)
    theta.append(theta[i] + dt*thetapoint[i])
```

rq : Face à une situation où la méthode d'Euler n'est pas jugée suffisamment précise, il faut soit diminuer le pas de temps (si le temps de calcul reste raisonnable), soit utiliser une méthode d'ordre supérieure.

3.2 Utilisation de `odeint`

Obsolescence de `odeint`

rq : Le programme de PCSI indique « Utiliser la fonction `odeint` de la bibliothèque `scipy.integrate` (sa spécification étant fournie) ». Mais cette fonction va bientôt devenir obsolète au profit de `solve_ivp` qui comporte plus de fonctionnalités. Pour nos exemples simples, on peut indifféremment choisir une de ces deux fonctions, seul l'ordre des arguments est différent mais l'énoncé le précisera.

Principe de `odeint`

L'utilisation de la fonction `odeint` nécessite d'écrire le système différentiel sous la forme $\dot{X} = F(X,t)$. Pour une ED d'ordre 1, X est la fonction scalaire à déterminer. Mais pour un système différentiel, X est un vecteur dont les composantes sont les fonctions à déterminer.
ex : Pour notre problème : $X = [\dot{\theta}, \theta]$, et $F(X,t) = [-\sin(\theta)\omega_0^2, \dot{\theta}]$.

La procédure générale est :

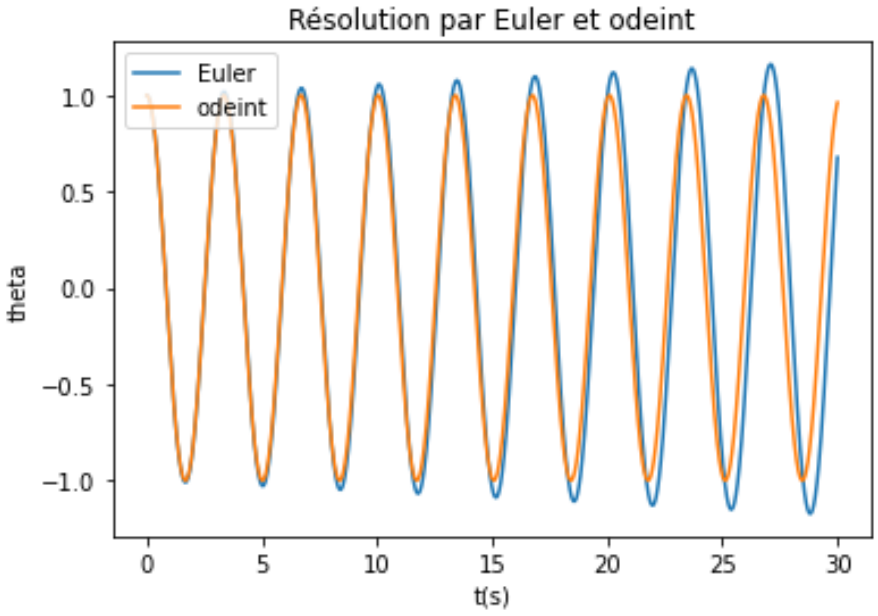


FIGURE 3 – Résolution du pendule pesant par méthode d'Euler et `odeint`. On observe que l'amplitude n'est pas constante par la méthode d'Euler, elle peut être jugée ici trop peu précise.

⋆ donner la condition initiale `CI` sur le vecteur X,
⋆ définir une fonction numérique `F` prenant en argument `X` et `t` (liste des instants),
⋆ appeler `odeint(F, CI, t)` qui renvoie un tableau dont le nombre de lignes est de même taille que `t` et le nombre de colonnes est le nombre de composantes de X.
ex : Par exemple ici, avec `solution = odeint(F, CI, t)`, appeler `solution[:,0]` donne accès à la liste des valeur de $\dot{\theta}$ et `solution[:,1]` donne accès à la liste des valeur de θ.

Code et comparaison à méthode d'Euler

```
## Resolution par odeint, X = [thetapoint, theta]
CI = [thetapoint0,theta0]          # Condition initiale sur X = [thetapoint, theta]
def F(X,t):
    return [-np.sin(X[1])*w0**2,X[0]]    # D'apres l'ED

solution = odeint(F, CI, t)

plt.plot(t,theta, label = 'Euler')
plt.plot(t,solution[:,1], label = 'odeint')
plt.title('Resolution par Euler et odeint')
plt.xlabel('t(s)')
plt.ylabel('theta')
plt.legend(loc = "upper left")
plt.show()
```

à retenir : La résolution numérique d'un système différentiel sera quasiment toujours plus précise et rapide par la fonciton `odeint` (ou `solve_ivp`) que par la méthode d'Euler. Sauf exception, n'utilisez pas la méthode d'Euler en TIPE !

3.3 Mise en évidence du non-isochronisme des oscillations

Dans le cadre des petits angles, on obtient une période $T_0 = 2\pi/\omega_0$ indépendante de l'amplitude. Mais expérimentalement, on observe que la période augmente avec l'amplitude.

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.integrate

w0 = 7                          # pulsation propre en rad/s
t = np.linspace(0,5,5000)       # liste des instants (prendre qq periodes)
```

```python

# Avec X = [thetapoint, theta]
def F(X,t):          # Fonction definissant le systeme d'equa diff
    return [-w0**2*np.sin(X[1]),X[0]]


plt.figure()
plt.xlabel('Temps (s)')
plt.ylabel('Theta')

# lacher sans vitesse initiale a differents angles
for theta0 in [10, 30, 60, 90, 120, 178]:    # theta ici en degres
    CI = [0,theta0*np.pi/180]                # CI [thetapoint en rad/s, theta converti en rad]
    solution = scipy.integrate.odeint(F,CI,t)      # Resolution du systeme differentiel
    plt.plot(t,solution[:,1]*180/np.pi,label='Theta(0)='+str(theta0))
plt.legend()
```

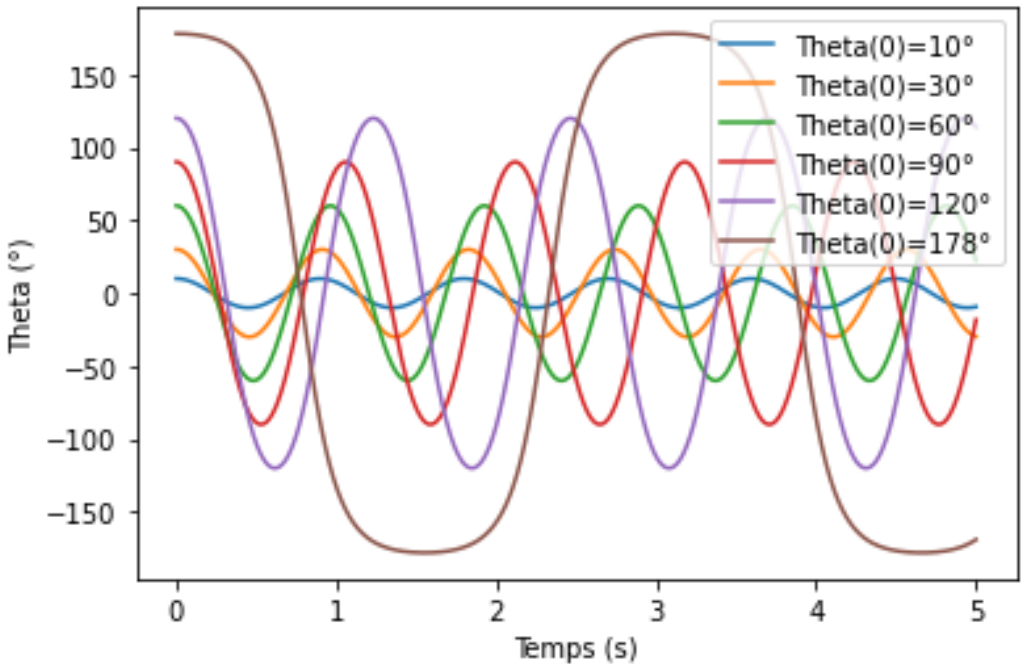


FIGURE 4 – Résolution du pendule pesant par `odeint` pour différentes amplitudes.

à retenir : Pour de petites amplitudes, l'évolution est bien sinusoïdale de période $2\pi/\omega_0$ (oscillateur harmonique). Mais plus l'amplitude augmente, plus la période augmente, et plus le signal se déforme, c'est dû au caractère non linéaire de l'équation différentielle (terme en sinus).

4 Calcul d'intégrale

→ Pendule pesant : *mettre en évidence le non isochronisme des oscillations. Mettre en oeuvre la méthode des rectangles pour calculer une valeur approchée d'une intégrale sur un segment.*

4.1 Rappels : formule des méthodes des rectangles et des trapèzes

Méthode des rectangles : $$I_{a,b}(f) = \int_a^b f(x)dx \simeq \frac{b-a}{n}\sum_{k=0}^{n-1} f(x_k).$$

Méthode des trapèzes :

$$I_{a,b}(f) \simeq \frac{b-a}{n}\sum_{k=0}^{n-1}\frac{f(x_k)+f(x_{k+1})}{2} = \frac{b-a}{n}\left(\sum_{k=0}^{n} f(x_k) - \frac{f(a)+f(b)}{2}\right).$$

4.2 Annexe sur la fonction quad du module scipy.integrate

Le module `scipy.integrate` contient une fonction nommée `quad` qui permet le calcul approché d'une intégrale.

⋆ Utilisation la plus simple :

• Trois arguments : `quad(f,a,b)` pour intégrer la fonction `f` entre `a` et `b`.

• Ce qui renvoie un couple `(I, e)` où `I` est une valeur approchée de l'intégrale $I_a^b(f) = \int_a^b f(x)dx$ et et `e` une estimation de l'erreur $|I(f) - \texttt{I}|$.

• Exemple de calcul de $\int_0^1 e^x dx = e - 1 \simeq 1,71828182846$:

```
>>> quad(np.exp,0,1)
(1.7182818284590453, 1.9076760487502457e-14)
```

• Si on ne s'intéresse qu'à la valeur $I_a^b(f)$, on peut n'appeler que le premier élément de `quad` (d'indice 0) en ajoutant `[0]` :

```
>>> quad(np.exp,0,1)[0]
1.7182818284590453
```

⋆ Utilisation pour une fonction à plusieurs arguments :

Si on veut calculer l'intégrale d'une fonction à plusieurs arguments $I(x) = \int_a^b g(x,t)dt$ pour x prenant la valeur x_0. Il faut d'abord définir `g` avec l'argument d'intégration en premier (t dans cet exemple). Puis utiliser l'option `args` de `quad` pour indiquer que le deuxième paramètre vaut `x0` :

```
def g(t,x): # on a mis t en premier
    return( ... expression de g(t,x) ... )

>>> quad(g, a, b,args=x0)[0]
```

4.3 Influence de l'amplitude sur la période du pendule

L'équation différentielle du pendule simple n'a pas de solution analytique. Mais on peut déterminer une expression théorique intégrale de la période $T(\theta_0)$ du pendule en fonction de l'amplitude θ_0 des oscillations et de la pulsation propre $\omega_0 = \sqrt{g/l}$:

$$T(\theta_0) = \frac{2\sqrt{2}}{\omega_0} \int_{\theta=0}^{\theta_0} \frac{d\theta}{\sqrt{\cos\theta - \cos\theta_0}}$$

Cette intégrale n'a pas d'expression analytique simple. On propose de la calculer numériquement avec `Python`. On prendra $\omega_0 = 3$ rad/s. On utilisera `np.pi`, `np.sqrt` et `np.cos`.

1. Écrire la conservation de l'énergie mécanique entre $\theta(t)$ et θ_0 puis effectuer une séparation des variables pour démontrer l'intégrale.
2. Compléter le code définissant la fonction à deux variables `integrande_pendule(theta,theta0)` qui correspond à la fonction à intégrer.
3. *Compléter le code.* Utiliser la fonction `quad` du module `scipy.integrate` pour calculer la période $T(\theta_0)$ des oscillations pour $\theta_0 = 0,1$ rad et pour $\theta_0 = 3$ rad et comparer ces résultats à la formule donnée par la linéarisation de l'équation différentielle $T(\theta_0) \simeq T_0 = 2\pi/\omega_0$.
4. *Compléter le code.* Tracer le graphe de $T(\theta_0)$ en fonction de l'amplitude $\theta_0 \in [0, \pi[$.

Il existe une formule approchée donnant une meilleure estimation de la période du pendule que la valeur de T_0. Il s'agit de la formule de Borda [2] :

$$T(\theta_0) \simeq T_0 \left(1 + \frac{\theta_0^2}{16}\right)$$

5. Compléter le graphe précédent avec la période donnée par la formule de Borda.
6. *Compléter le code.* Écrire une fonction `borda_juste(p)` qui retourne l'angle $\theta_{\max}$ tel que la formule de Borda donne le vrai résultat à moins de p pourcent près pour $\theta < \theta_{\max}$.

2. La démo est hors-programme. Pour info, les termes suivants sont $\frac{11}{3072}\theta_0^4 + \frac{173}{737280}\theta_0^6 + O(\theta_0^8)$.

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad

# Definition des ctes du pb
omega0 = 3.0                    # pulsation des petites oscillations
periode0 = 2*np.pi/omega0       # periode des petites oscillations


# Question 2 ##################################################################
def integrande_pendule(theta,theta0):       # definition de l'integrande
    return                                  # a completer

# Question 3 ##################################################################
periode1 =                                              # a completer
periode2 =                                              # a completer

# Question 4 ##################################################################
# Je choisis par exemple une centaine de points :  # a completer
THETA = np.linspace(
PERIODE=[]
for k in              :  # Exclure theta=0 !
    PERIODE.append(

plt.figure()
plt.plot(THETA[1:],PERIODE,label="Expression integrale")
plt.xlabel('amplitude (rad)')
plt.ylabel('periode (s)')

# Question 5 ##################################################################
plt.plot(THETA,periode0*(1+THETA**2/16),label="Approximation quadratique de Borda")
plt.plot([0,np.pi],[periode0,periode0],label="Approximation lineaire")
plt.xlim(0,np.pi)
plt.ylim(0,4*periode0)
plt.grid()
plt.title('Periode du pendule simple')
plt.legend(loc='upper left')
plt.show()

# Question 6 ##################################################################
def borda_juste(p): # a completer
```

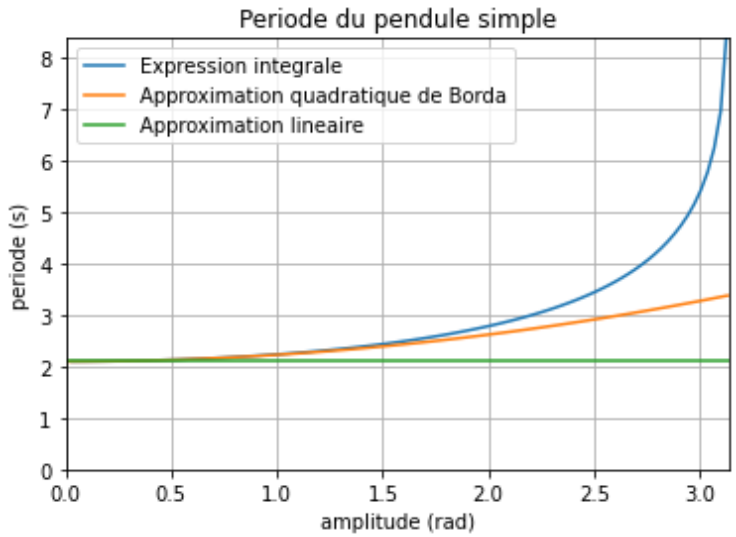


FIGURE 5 – Influence de l'amplitude sur la période par calcul approché d'intégrale.

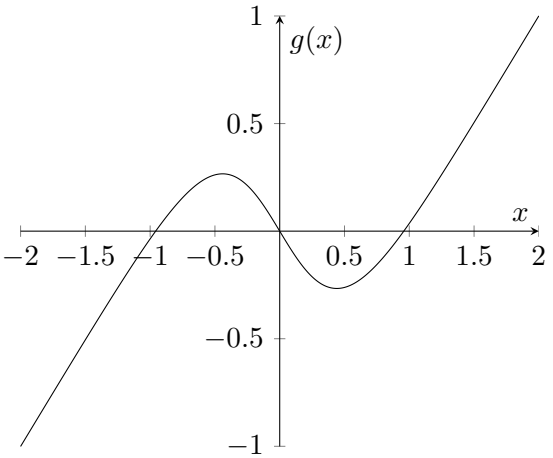
à retenir : Pour de petites amplitudes, l'évolution est bien de période $2\pi/\omega_0$ (oscillateur harmonique). Mais plus l'amplitude augmente, plus la période augmente. On observe même une divergence quand l'amplitude tend vers π.

5 Aimantation d'un ferromagnétique (dichotomie)

→ *Déterminer, en s'appuyant sur une représentation graphique, un intervalle adapté à la recherche numérique d'une racine par une méthode dichotomique. Mettre en oeuvre une méthode dichotomique afin de résoudre une équation avec une précision donnée. Utiliser la fonction* `bisect` *de la bibliothèque* `scipy.optimize` *(sa spécification étant fournie)..*

Un matériau ferromagnétique a la capacité de conserver son aimantation si la température est suffisamment basse, même en l'absence de champ magnétique extérieur, ce qui permet notamment d'en faire des aimants permanents. Dans le cadre d'un modèle simple, calculer l'aimantation du matériau revient à chercher les solutions de l'équation $x = \tanh(\alpha.x)$ où tanh est la fonction tangente hyperbolique, et α est un coefficient dépendant de la température. On remarque que $x = 0$ est toujours une solution, mais que deux solutions opposées existent si $\alpha > 1$. On se placera dans le cas $\alpha = 2$ (correspondant à une température donnée non précisée ici). Le problème est alors équivalent à trouver une racine de la fonction g suivante, dont on donne le graphe sur $[-2, 2]$:

$$g : x \mapsto x - \tanh(2x)$$



On remarque sur son graphe qu'elle possède trois racines : $x = 0$ et $x = \pm r$ avec $r > 0$. L'objectif de l'exercice est alors de déterminer numériquement la valeur de la racine strictement positive r.

On considérera avoir à disposition en `Python` l'instruction `tanh(x)` pour calculer numériquement $\tanh(x)$ (issue de la bibliothèque `numpy`).

La bibliothèque `scipy.optimize` contient la fonction `bisect` qui permet une recherche de racine par dichotomie. Par exemple `bisect(f,a,b)` renvoie une racine de f dans l'intervalle $[a, b]$ (si $f(a)$ et $f(b)$ vérifie la condition habituelle demandée dans l'exercice suivant). On peut aussi proposer des arguments supplémentaires comme un nombre maximal d'itérations.

1. Écrire un code pour définir la fonction numérique `g` à un paramètre `x` qui code les valeurs de la fonction g.
2. Quelles sont les conditions d'applications d'une recherche de racine par dichotomie ? Proposer un intervalle adapté pour la détermination par dichotomie de r.
3. Sans la fonction `bisect`, écrire un code qui détermine une valeur approchée à 10^{-8} près de la racine r par dichotomie. Il faudra commenter le code par une phrase précisant quelle variable contient la solution approchée à la fin du calcul numérique.
4. Dans une recherche dichotomique, l'intervalle de recherche est réduit à chaque itération d'une boucle. En partant d'un intervalle initial de longueur 1, quelle est la taille de l'intervalle de recherche après n itérations ?
5. En déduire combien de passages n dans la boucle sont nécessaires pour obtenir une valeur approchée à 10^{-8} près ? *[n est un entier. Même sans calculatrice, essayer de donner au moins un ordre de grandeur de cet entier (à quelques unités près).]*

6 Trajectoire dans un champ de force centrale

→ Point matériel soumis à un champ de force centrale conservatif. Conservation de l'énergie mécanique. Énergie potentielle effective. État lié et état de diffusion. : *obtenir des trajectoires d'un point matériel soumis à un champ de*

force centrale conservatif. [3]

Intérêt d'utiliser une méthode numérique dans ce cas

⋆ Faire varier facilement les différents paramètres pour observer leur influence.
⋆ Pour une force newtonienne [4] : la résolution analytique n'est pas exigible en niveau CPGE [5].
⋆ Pour d'autres exemples de force centrale, la résolution analytique n'est pas possible.
⋆ Rappel : ne pas utiliser la méthode d'Euler explicite, les trajectoires obtenues risqueraient de ne pas boucler même pour un état lié attendu !

6.1 Position du problème

Considérons la Terre modélisée par une sphère de rayon $R_T = 6370$ kg et masse $M_T = 5,96.10^{24}$ kg. On étudie un satellite repéré par ses coordonnées [6] $x(t)$ et $y(t)$. On note le vecteur $X(t) = (x(t), y(t), \dot{x}(t), \dot{y}(t))$. Considérons les conditions initiales $X_0 = (x_0, y_0, \dot{x}_0, \dot{y}_0) = (0, 65000 \text{ km}, 1200 \text{ m.s}^{-1}, 0)$. On donne la constante gravitationnelle $\mathcal{G} = 6,67.10^{-11}$ m^3.kg^{-1}.s^{-2}. On veut déterminer la trajectoire du satellite.

Rappel de la procédure générale :
⋆ donner la condition initiale `CI` sur le vecteur X,
⋆ définir une fonction numérique `F` prenant en argument `X` et `t` (liste des instants),
⋆ appeler `odeint(F, CI, t)` qui renvoie un tableau `X_t` dont le nombre de lignes est de même taille que `t` et le nombre de colonnes est le nombre de composantes de X.

6.2 Résolution numérique

1. Dans le code suivant, compléter la ligne correspondant à la définition de la fonction `F` qui renvoie la dérivée de `X` à partir des équations différentielles.
2. Dans le code suivant, compléter la ligne correspondant au calcul de `X_t`.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
#------------------Constantes------------------------------
# Constante de gravitation
G = 6.67e-11
# Masse Terre
MT = 5.96e24
# Pour expression de la force en mK/r**2
K = -G*MT
#-----------------Parametres a renseigner-----------------
# Vitesse initiale (m/s)
v0x = 1200
v0y = 0
# Position x initiale (m)
x0 = 0
y0 = -65000000
# Temps
t = np.linspace(0,1000000,1000000)
#----------------Vecteur conditions initiales sur X-------------
# Vecteur conditions initiales sur X
X0 = np.array([x0,y0,v0x,v0y])

# Fonction qui renvoie dX/dt a partir des equa diff
def F(X,t):
    return np.array( ____________________________________          # a completer

# Resolution du systeme d'equa diff par odeint

```

3. Rappel : Une force s'appliquant au point matériel M est dite centrale lorsque sa droite d'action passe constamment par un point fixe O du référentiel d'étude. La force est alors $\overrightarrow{F} = F(r, \theta, \varphi)\overrightarrow{u_r}$ en sphérique avec O au centre.
4. Rappel : une force est dite « newtonienne » si $\overrightarrow{F} = \frac{k}{r^2}\overrightarrow{u_r}$ avec k constante.
5. Mais accessible si l'énoncé guide, cf composition de physique, agrégation 2018, sujet difficulté type ENS.
6. Comme le mouvement est plan pour force centrale, on peut utiliser (x, y) ou (r, θ) en cylindrique.

```
X_t = ____________________________          # a completer

# Conversion facultative en km
x_km = X_t[:,0]/1000
y_km = X_t[:,1]/1000
# Graphe de la trajectoire
plt.figure()
plt.axis('equal')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Trajectoire')
circle1 = plt.Circle((0, 0), 6370, color='r')  # Pour visualiser la Terre representee par une sphere
plt.gca().add_patch(circle1)
plt.grid("on")
plt.plot(x_km,y_km,'k')
```

6.3 Interprétation des résultats

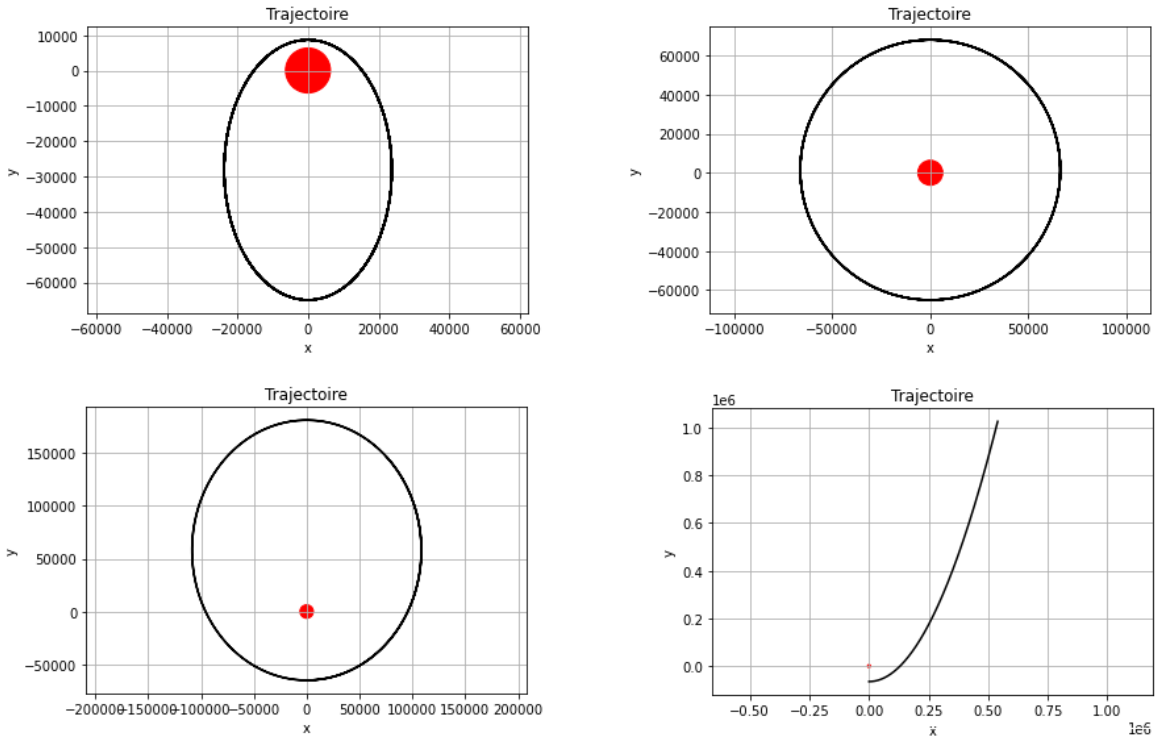


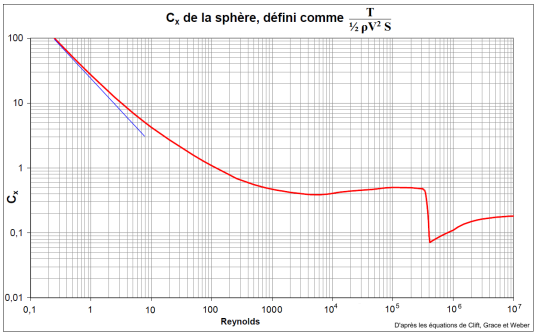
FIGURE 6 – Trajectoires obtenues pour des vitesses initiales respectives $\dot{x}_0 = 1200, 2500, 3000, 3500$ m/s. Dans le dernier cas, la trajectoire n'est pas bornée, même en calculant sur une plus longue durée.

3. Sans faire de calcul, que donnerait le code si on avait pris $\dot{x}_0 = 0$?
4. Proposer un calcul à effectuer pour interpréter la trajectoire circulaire du deuxième graphe.
5. Proposer un calcul à effectuer pour interpréter la trajectoire non bornée du dernier graphe.

7 Évolution de la température et la pression dans l'atmosphère

→ Statique dans le champ de pesanteur uniforme : *étudier les variations de température et de pression dans l'atmosphère.*

7.1 Intérêt d'utiliser une méthode numérique dans ce cas



Par exemple, considérons la chute d'une sphère rigide de rayon R dans un fluide de masse volumique μ. La force subie est de norme $F = \frac{1}{2}.\mu.\pi R^2.C_x(v).v^2$ où le paramètre C_x dépend de la vitesse v, cf graphe ci-contre et chapitre *MF2*. La résolution analytique est possible dans les cas limites où une expression explicite de C_x en fonction de v est connue. Mais dans le cas général, on n'a accès qu'à une liste des valeurs de C_x. La résolution analytique est alors impossible mais la résolution numérique n'est presque pas plus difficile que dans le cas d'un C_x constant !

De manière générale, il est fréquent qu'un système réel soit modélisé par une équation différentielle dont un des paramètres n'est pas constant mais ses valeurs sont répertoriées dans un tableau.

7.2 Position du problème

Considérons l'atmosphère terrestre considérée comme un gaz parfait (masse molaire $M_{\text{air}} = 29.10^{-3}$ kg.mol^{-1}) au repos dans le référentiel terrestre dans le champ de pesanteur uniforme $g = 9,81$ m.s^{-2}. On fixe les valeurs de la température et de la pression au niveau du sol (en $z = 0$) respectivement à $T_{\text{sol}} = 288$ K et $P_{\text{sol}} = 1,013.10^5$ Pa. On cherche l'évolution de la température $T(z)$ et la pression $P(z)$ en fonction de l'altitude z ascendante.

L'équation de la statique des fluides donne $\frac{dP}{dz} = -\frac{M_{\text{air}}g}{RT}P$ qu'on sait résoudre analytiquement dans le cas du modèle d'atmosphère isotherme (question de cours) ou alors si le gradient de température $dT/dz = k$ est constant (à essayer en exo).

Couche atmosphérique	Altitude de la base (en km)	Gradient thermique vertical (en K/km)
Troposphère	0	− 6.5
Tropopause	11	0
Stratosphère	20	+ 1.0
Stratosphère	32	+ 2.8
Stratopause	47	0
Mesosphère	51	− 2.8
Mesosphère	71	− 2.0
Mesopause	85	-

On souhaite ici aller plus loin pour déterminer numériquement la loi de variation de la pression atmosphérique avec l'altitude z dans le cadre du modèle ISA. Dans le cadre du modèle ISA (International Standard Atmosphere), l'atmosphère est divisée en différentes couches, au sein desquelles la température est supposée suivre une loi affine [7]. La valeur du gradient vertical de température $k_{\text{ISA}}(z)$ dans chacune de ces couches est normalisée, cf tableau ci-contre. Le système différentiel à résoudre est alors :

$$\begin{cases} \frac{dT}{dz}(z) = k_{\text{ISA}}(z) \\ \frac{dP}{dz}(z) = -\frac{M_{\text{air}}g}{RT(z)} \cdot P(z) \end{cases}$$

7.3 Résolution numérique

```
## Importation des bibliotheques
import numpy as np                  # pour la manipulation des tableaux
import matplotlib.pyplot as plt     # pour les representations graphiques
from scipy.integrate import odeint  # pour la resolution des equations differentielles
## Definition des constantes du probleme
g = 9.81                            # acceleration de la pesanteur (en m/s^2)
Mair = 29e-3                        # masse molaire de l'air (en kg/mol)
R = 8.314                           # constante du gaz parfait (en J/K/mol)
Tsol = 288                          # temperature de l'atmosphere au niveau du sol (en K)
Psol = 1.013e5                      # pression de l'atmosphere au niveau du sol (en Pa)
## Definition du gradient thermique vertical selon le modele ISA
def kISA(z):
    """ z est l'altitude en metres. La fonction renvoie la valeur du gradient thermique
    vertical a l'altitude z (en K/m). """
    if 0 <= z < 11e3: return -6.5e-3
    elif z < 20e3: return 0
    elif z < 32e3: return 1.0e-3
    elif z < 47e3: return 2.8e-3
    elif z < 51e3: return 0
    elif z < 71e3: return -2.8e-3
    elif z < 85e3: return -2.0e-3
    else: return None

```

7. Wikipédia, *Atmosphère normalisée*, page consultée le 22 mars 2021.

```python
## Definition du systeme differentiel a resoudre
def systDiff(TP,z):
    """
    TP designe le vecteur inconnu de dimension 2 (TP[0] : temperature ; TP[1] : pression) ;
    z designe l'altitude.
    La fonction renvoie respectivement la derivee de la temperature et la derivee de
    la pression a l'altitude z.
    """
    # Lois prevues par le modele theorique
    dT = kISA(z)                    # derivee verticale de la temperature
    dP = - Mair*g/R*TP[1]/TP[0]     # derivee verticale de la pression
    return [dT, dP]

## Definition des conditions aux limites
CAL = [Tsol, Psol]

## Definition de l'ensemble des valeurs de z pour lesquelles on cherche la solution
## numerique approchee du systeme differentiel
z = np.linspace(0, 85e3, 10000)     # on choisit 10000 points regulierement espaces entre 0
                                    # et 85 km d'altitude

## La resolution en elle-meme

TP = odeint(systDiff, CAL, z)
T = TP[:,0]                         # extraction des valeurs de la temperature
P = TP[:,1]                         # extraction des valeurs de la pression

## Representation graphique des resultats
def PisoT(z, T0 = 288):             # pour comparaison
    """ Calcule la valeur de P a l'altitude z selon le modele isotherme.
    Par defaut, la temperature est fixee a 288 K. """
    return Psol*np.exp(-Mair*g*z/(R*T0))

plt.figure(figsize=(13,6))
plt.subplot(1,2,1)                  # graphique de gauche
plt.title("Evolution de la temperature avec l'altitude")
plt.plot(T, z*1e-3, 'b-')
plt.xlim(180,300), plt.xlabel(r"$T$ (en K)")
plt.ylim(0,85), plt.ylabel(r"$z$ (en km)")
plt.grid()
plt.subplot(1,2,2)                  # graphique de droite
plt.title("Evolution de la pression avec l'altitude")
plt.semilogx(P*1e-5, z*1e-3, 'b-', label = "Modele ISA")
plt.semilogx(PisoT(z)*1e-5, z*1e-3,'m:', label = r"Modele isotherme ($T=T_{\rm sol}$)")
plt.xlim(1e-6,1), plt.xlabel(r"$P$ (en bar)")
plt.ylim(0,85), plt.ylabel(r"$z$ (en km)")
plt.legend(loc=0)
plt.grid(which = 'both')
plt.show()

plt.figure()                        # comparaison des modeles ISA et isotherme
plt.plot(z*1e-3, abs(P-PisoT(z))/P, 'y-')
plt.xlim(0,20), plt.xlabel(r"$z$ (en km)")
plt.ylim(0,1), plt.ylabel(r"ecart relatif : $\dfrac{\vert P-P_{\rm iso}\vert}{P}$")
plt.grid()
plt.show()
```

7.4 Interprétation des résultats

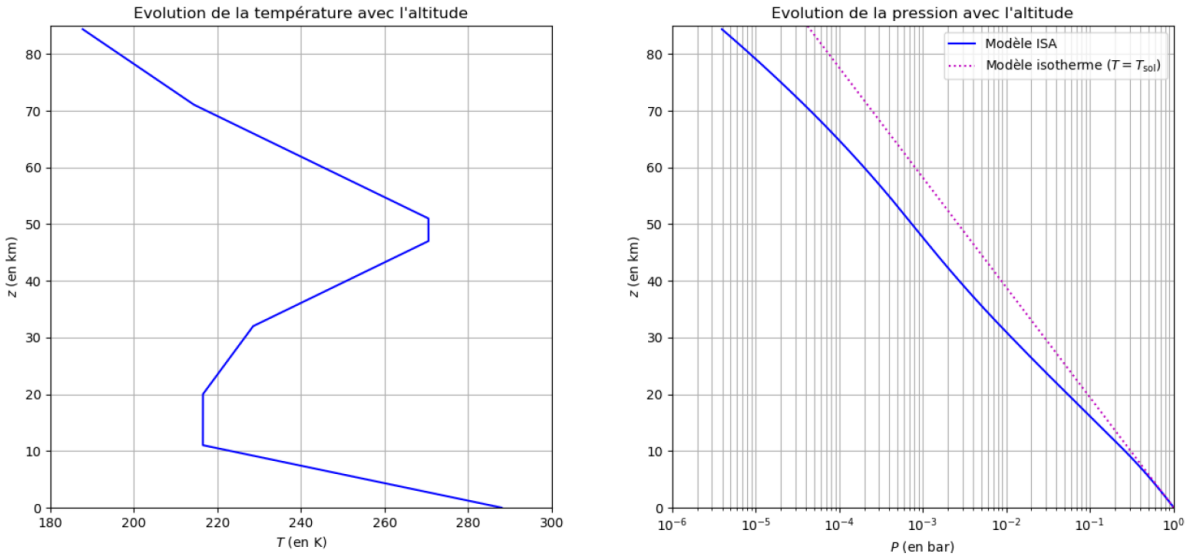


FIGURE – Résultats de la résolution numérique du système différentiel entre 0 et 85 km d'altitude. À gauche : Évolution de la température T (en abscisse) en fonction de l'altitude z (en ordonnée). À droite : Évolution de la pression P (en abscisse ; échelle semi-logarithmique) en fonction de l'altitude z (en ordonnée).

Comme on pouvait s'y attendre, le modèle de l'atmosphère isotherme donne un champ de pression comparable au modèle ISA à basse altitude. En revanche, au-delà de quelques kilomètres d'altitude, l'écart entre les deux modèles devient conséquent.

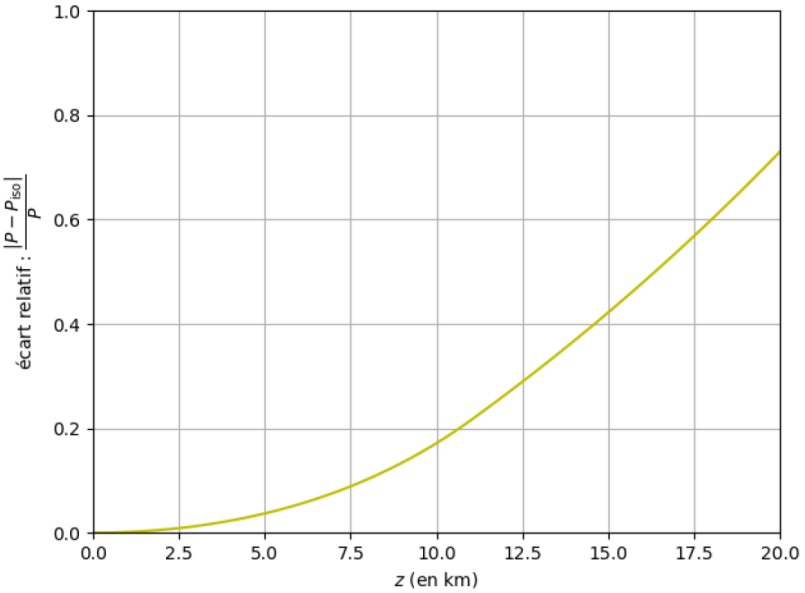


FIGURE – Évolution avec l'altitude de l'écart entre les pressions déterminées par les modèles ISA et isotherme (avec $T = T_{sol} = 288$ K) dans la troposhère et la tropopause.